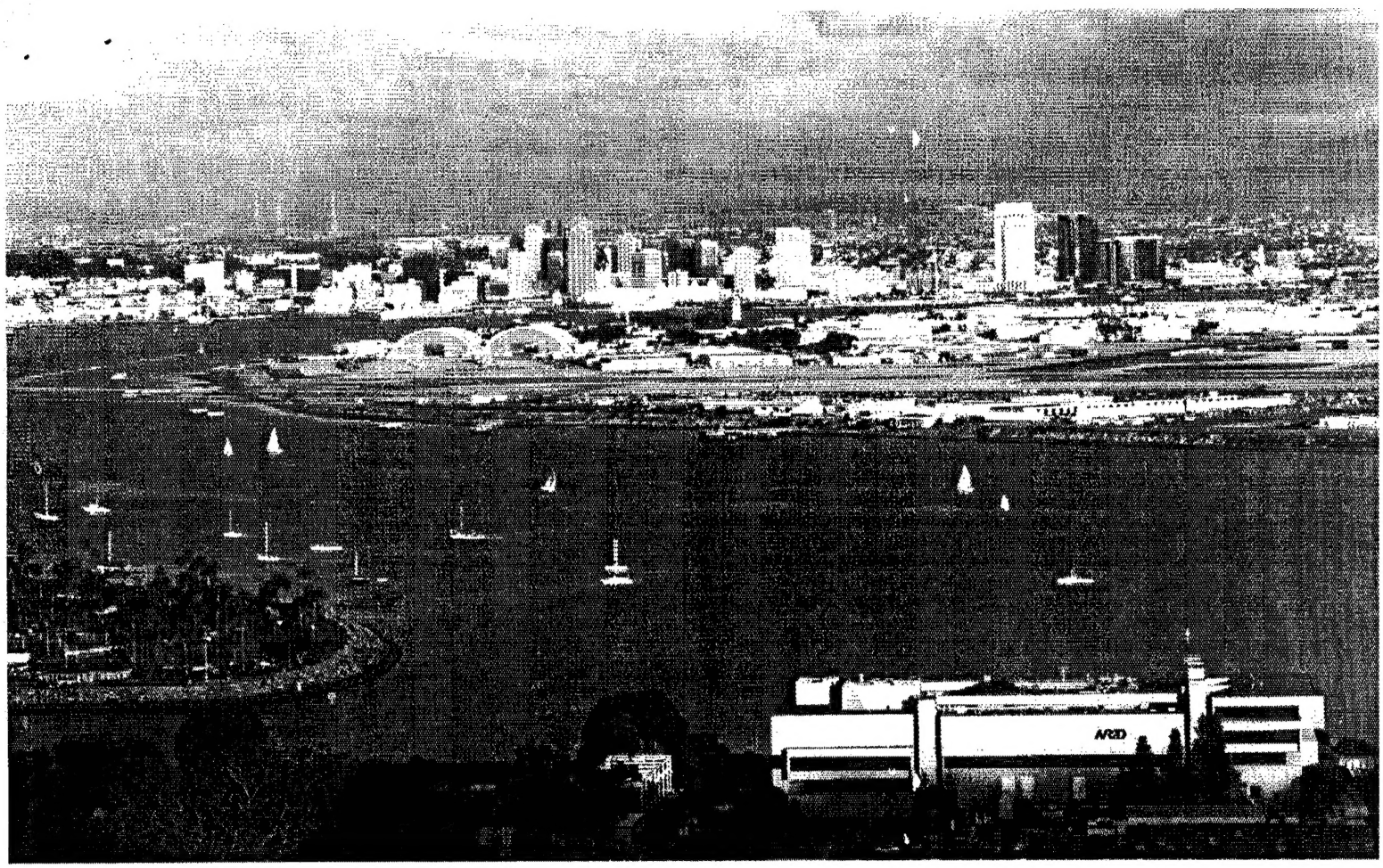


An Ordering Algorithm for Exposing Parallelism in Sparse Symmetric Matrices

DTIC
ELECTE
OCT 0 6 1995
S D
G

Aram K. Kevorkian

TR 1697
May 1995

Naval Command, Control and
Ocean Surveillance Center
RDT&E Division

San Diego, CA
92152-5001

19951004 157

DTIC QUALITY INSPECTED 6

Authorized for public release; distribution is unlimited.

Authorized for public release; distribution is unlimied.

Technical Report 1697
May 1995

An Ordering Algorithm for Exposing Parallelism in Sparse Symmetric Matrices

Aram K. Kevorkian

Accesion For	
NTIS CRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152–5001

K. E. EVANS, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

This report was sponsored by the Office of Chief of Naval Research and performed under the NRaD Independent Research Program, project ZW62.

Released by
R. H. Hearn, Head
Senior Technical Staff

Under authority of
P. M. Reeves, Head
Analysis and Simulation
Division

ACKNOWLEDGMENTS

The author would like to express his sincere thanks to Dr. Michael Saunders for providing the sparse symmetric matrices from the linear and quadratic programming applications, and to Curt Goodhart at NRaD for his help in getting the Harwell–Boeing sparse matrix problems into Matlab. The author would also like to thank Dr. Michael Heath and the referees for constructive and helpful suggestions.

LH

EXECUTIVE SUMMARY

OBJECTIVE

Given a sparse symmetric matrix M, we develop an ordering algorithm to find a permutation matrix P so that parallelism inherent in M is fully exposed in the matrix PMP^T.

RESULTS

An ordering algorithm with the following key stages was developed. First, compute in the undirected graph G = (V, E) of M a set of vertices S^* such that the induced subgraph $G(V - S^*)$ contains parallel regions inherent in M. This stage gives rise to a block diagonal matrix A such that each diagonal block is a full matrix in M. Second, factor block diagonal matrix A symbolically and compute the symbolic form of the Schur complement of A in M. Third, replace original matrix M by the symbolic Schur complement, and repeat the process until the symbolic Schur complement is a full matrix. By a property of the set S^*, the ordering algorithm takes advantage of all principal submatrices of M that do not produce fill-in in any part of M when symbolically factored.

CONCLUSIONS

For a sparse symmetric matrix M, we have presented an ordering algorithm to compute a permutation matrix P so that parallelism inherent in M is fully exposed in PMP^T. The application of the new parallel ordering algorithm to a large set of sparse matrices taken from the Harwell–Boeing sparse matrix collection and industrial linear and quadratic programming problems show that this algorithm is an effective tool for exposing parallelism in arbitrary sparse symmetric matrices. Our experiments also show that the new parallel ordering algorithm compares favorably with a highly refined implementation of the minimum degree method in keeping the number of fill-ins and elimination tree heights small.

CONTENTS

1 INTRODUCTION 1

2 NOTATION 2

3 A TOOL FOR EXPOSING PARALLELISM IN SPARSE SYMMETRIC MATRICES 3

4 A PARALLEL ORDERING ALGORITHM DERIVED FROM VERTEX PARTITION Π^* 6

5 IMPLEMENTATION OF THE PARALLEL ORDERING ALGORITHM 8
- 5.1 COMPUTING THE SET OF VERTICES S 8
- 5.2 COMPUTING CONNECTED COMPONENTS OF G(V – S) 8
- 5.3 CLASSIFYING THE CLIQUES IN G(V – S) 9
- 5.4 COMPUTING INDEPENDENT CLIQUES IN G(V – S) 13
- 5.5 COMPUTING THE SYMBOLIC SCHUR COMPLEMENT 13

6 COMPUTATIONAL RESULTS, COMPARISONS, AND COMPLEXITY 16

7 CONCLUSIONS 25

8 REFERENCES 25

Figures

1 Condensation of G with respect to vertex partition P* 6

2 Sparsity structure of R^T+R for problem scfxm1 18

3 Sparsity structure of R^T+R for problem shell 18

4 Sparsity structure of R^T+R for problem pores_3 20

5 Sparsity structure of R^T+R for problem 685_bus 20

6 Number of parallel tasks per iteration in percent of number of parallel tasks at first iteration 24

Tables

1 Examples from linear and quadratic programming applications 17

2 Examples from Harwell–Boeing sparse matrix collection 19

3 Size of Schur complement in percentage of original matrix size 22

4 Execution times at completion of first five iterations in percentage of overall time 23

1. INTRODUCTION

A central problem in the solution of large sparse symmetric positive definite systems of equations

$$Mx = b$$

is finding a permutation matrix P such that PMP^T has a sparse Cholesky factor. The pivotal importance of the permutation matrix P in the total solution of the symmetric system of equations $Mx = b$ has led to several highly successful and widely used sparse matrix ordering methods. These include the nested dissection method (George & Liu, 1981, minimum degree ordering (George & Liu, 1981, 1989; Liu, 1985) and band and envelope reduction methods (George & Liu, 1981).

With the arrival of parallel architecture machines in the mainstream of advanced scientific computing, the main objective of permutation matrix P has been expanded so that the Cholesky factor of PMP^T is not only sparse but also suitable for parallel computation. The most popular notion used to date for the parallel ordering problem is that of the elimination tree associated with a Cholesky factor (Liu, 1986; Schreiber, 1982). The key role of the elimination tree model in the parallel ordering problem is to find a sparse Cholesky factor whose elimination tree has small height. Liu (1989) has described a two-stage method that accomplishes this objective by combining minimum degree and nested dissection orderings. Pothen, Simon, and Wang (1992) have recently reported a two-stage method (Liu, 1989) that accomplishes this task by combining minimum degree and nested dissection orderings. Pothen, Simon, and Wang (1992) have recently reported a spectral nested dissection algorithm that recursively uses a spectral separator algorithm to compute parallel orderings. A comprehensive survey covering parallel algorithms for sparse matrix computations through 1990 is given by Heath, Ng, and Peyton (1991).

In this report, we present a new ordering algorithm that recursively uses a four-stage parallelization tool to compute a permutation matrix P such that PMP^T has a sparse Cholesky factor and is suitable for parallel computation. Letting $G = (V, E)$ denote the undirected graph of the sparse symmetric matrix M, the four key stages of the parallelization tool are as follows:

1. Compute the set of vertices $S = \{v \in V \mid \exists\ (v,w) \in E \text{ with } \deg_G v > \deg_G w\}$;
2. Compute connected components of induced subgraph $G(V - S)$;
3. Classify the clique connected components of $G(V - S)$;
4. Compute independent cliques in nonclique connected components of $G(V - S)$.

Through these four stages, we obtain a set of vertices S^*, where $S \subseteq S^*$, such that the induced subgraph $G(V - S^*)$ contains parallel regions inherent in M. In matrix terms, the set of vertices S^* gives rise to a block diagonal matrix A such that each diagonal block is a full matrix in M. At this point, block A is factored symbolically to compute the symbolic form of the Schur complement of A in M. Lastly, the original matrix M is replaced by the symbolic Schur complement, and the process is repeated until the symbolic Schur complement is a full matrix.

The construction of the vertex set S in stage 1 forms the most novel and crucial piece of the parallelization tool. To highlight a key property of the set S, consider any clique in G with vertex set U, and suppose we partition U into two disjunct parts U' and U'' such that $U' = \{u \mid \deg_G u \le \deg_G v \text{ for all } v \in U\}$. We call the set of vertices U' the core of clique $G(U)$, and for the special case where a vertex u in U' satisfies the equality $\deg_G u = |U| - 1$, we call U' the interior of $G(U)$ and the induced subgraph $G(U')$ an interior clique. The matrix interpretation of an interior clique makes this subgraph

of G of some interest. To highlight this, let G(U) be any interior clique in G and let **A** be a principal submatrix of M corresponding to G(U). Then the symbolic factorization of submatrix **A** does not produce fill-in in any part of matrix M.

As a corollary to a decomposition result, we show that every interior clique in G is a connected component of the induced subgraph G(V – S), and so the first two stages of the parallelization tool isolate every principal submatrix that preserves sparsity in the process of symbolic factorization. Stage 3 deals with the connected components of G(V – S) that are cliques, while stage 4 deals with the connected components that are not cliques. Stage 3 classifies the noninterior cliques in G by using an interior clique as the clique of choice. Stage 4 exploits the underlying structure of a non-clique connected component. The total time required by the four-stage parallelization tool is proportional to the number of vertices and number of edges of G.

This report is organized as follows. In section 2, we cover the necessary graph-theoretic notation. In section 3, we give a concise formulation of the four-stage parallelization tool. In section 4, we formulate a new parallel ordering algorithm that recursively employs the four-stage parallelization tool until no sparsity is left for exploitation. In section 5, we give high-level implementations of the key procedures used in the new ordering algorithm. In section 6, we evaluate the time complexity of the new ordering algorithm and compare it with two commonly used ordering methods implemented in the linear algebra package Matlab (The Mathworks, 1990). The benchmark for the comparisons comprises a set of 21 sparse symmetric matrices drawn from the Harwell–Boeing sparse matrix collection (Duff, Grimes, & Lewis, 1989) and from applications of linear and quadratic programming to industrial problems (Saunders, M. A., private communication, 1992–1993). Conclusions are presented in section 7.

2. NOTATION

A graph G = (V, E) consists of a finite, nonempty set of vertices V and a set of edges E. If the edges are ordered pairs (u, v) of vertices, G is said to be directed. If the edges are unordered pairs of vertices, also denoted by (u, v), G is said to be undirected. All graphs in this work are assumed to be undirected and connected. For a subset U of the vertex set V, the induced subgraph G(U) of G is the subgraph G(U) = (U, E(U)) where

$$E(U) = \{(u,v) \in E \mid u, v \in U\}.$$

A set of vertices S is called a separator of G if G(V – S) is not a connected graph.

In a graph G = (V, E), a vertex v is said to be adjacent to another vertex w if (v, w) is an edge in E. The set

$$adj_G v = \{w \in V - \{v\} \mid (v, w) \in E\}$$

denotes the set of vertices adjacent to v. We call the set $adj_G v$ the adjacency of v in G. The degree of vertex v, denoted by $deg_G v$, is the number of vertices adjacent to v and so we have $deg_G v = |adj_G v|$. An induced subgraph G(U) of G is called a clique if each vertex in U is adjacent to every other vertex in U. A clique is maximal if it is not a proper subgraph of another clique. For any graph G = (V, E), a vertex partition is called a clique partition if each element of the partition induces a clique. A vertex v is called simplicial (Dirac, 1961; Lekkerkerker & Boland, 1962) if the subgraph of G induced by the adjacency of v is a clique.

For any n-by-n structurally symmetric matrix M, there exists an undirected graph $G = (V, E)$ such that vertex v_i in V represents row i of M, and the edge (v_i, v_j) is in E if and only if the element in the (i, j) location of M is nonzero for all $i \neq j$. Another representation for graph G is by means of vertex partitions. For any vertex partition $\Pi = (V_1, V_2, \ldots, V_k)$ in G, $G_\Pi = (V_\Pi, E_\Pi)$ is a graph such that each element V_i of the partition is a vertex in V_Π, and the pair (V_i, V_j) is an edge in E_Π if and only if a vertex in the set V_i is adjacent to a vertex in the set V_j. George and Liu (1981) call G_Π a quotient graph of G with respect to Π.

Suppose $G = (V, E)$ is the undirected graph of the n-by-n symmetric matrix M, and let v denote the vertex in V representing the ith row of M. Then the set of edges defined by

$$\mathrm{def}_G v = \{(u,w) \mid u, w \in \mathrm{adj}_G\, v, (u,w) \notin E, u \neq w\}$$

corresponds exactly to the fill-ins produced when the ith component of the vector x in the system of equations $Mx = b$ is eliminated from the original system (assuming no cancellation of nonzero elements). The set of edges $\mathrm{def}_G v$ is called the deficiency of v in G (Rose, Tarjan, & Lueker, 1976). The graph $G_v = (V - \{v\}, E(V - \{v\}) \cup \mathrm{def}_G v\}$, obtained by adding the deficiency $\mathrm{def}_G v$ to the induced subgraph $G(V - \{v\})$, is precisely the undirected graph of the $(n - 1)$ by $(n - 1)$ coefficient matrix in the reduced system of $n - 1$ equations. The graph G_v is called the v-elimination graph of G (Rose, Tarjan, & Lueker, 1976). The first graph-theoretic characterization of Gaussian elimination on symmetric matrices is due to Parter (1961).

3. A TOOL FOR EXPOSING PARALLELISM IN SPARSE SYMMETRIC MATRICES

We begin by developing a graph-theoretic method for exposing parallelism in general sparse symmetric matrices.

For any set of vertices U in $G = (V, E)$, let $\delta(U)$ denote the minimum degree among the vertices in U or

$$\delta(U) = \min_{u \in U} \deg_G u\ .$$

Using the parameter $\delta(U)$, we define the core of a clique G(U) to be the set of vertices cor(U) given by

$$\mathrm{cor}(U) = \{u \in U \mid \deg_G u = \delta(U)\}.$$

The core of a clique G(U) thus partitions the vertex set U into two disjunct parts, cor(U) and U – cor(U), so that all vertices with local minimum degree $\delta(U)$ are contained in cor(U). The significance of this partition will become apparent when we elaborate on the special case where $\delta(U)$ attains its smallest value in G.

Since G(U) is a clique, we have $\delta(U) \geq |U| - 1$, and so the smallest value $\delta(U)$ can take in G is $|U| - 1$. To accommodate this special case, let us define the interior of the clique G(U) to be the set of vertices int(U) given by

$$\mathrm{int}(U) = \{u \in U \mid \deg_G u = |U| - 1\}.$$

The core and interior of a clique G(U) are thus identical if and only if $\delta(U) = |U| - 1$. If this equality does not hold, then G(U) is a clique with an empty interior and a nonempty core. In the case that G(U) is a clique with a nonempty interior, we call the subgraph induced by the interior of the clique G(U) an interior clique.

The interior of a clique has properties that are ideally suited for preserving sparsity in sparse matrix computations and for exposing parallelism in matrices with regular and irregular structures. We will briefly outline these two properties and then present the main result in this work.

Let G(U) be any clique in G = (V, E) with a nonempty interior and let v be any vertex in int(U). Then $\deg_G v = |U| - 1$, and so all vertices adjacent to v are in U since G(U) is a clique. Thus, v is a simplicial vertex, and so an interior clique is simply a clique in which every vertex is simplicial. From this property of an interior clique, it follows that if M is a symmetric matrix with the undirected graph G, then the symbolic factorization of the submatrix of M corresponding to G(int(U)) will not give rise to fill-in in any part of matrix M. In other words, the sparsity of M is fully preserved when a submatrix of M corresponding to an interior clique in G is symbolically factored. This highlights the first of two key properties exploited in our parallel ordering algorithm.

Since all vertices adjacent to any vertex v in int(U) are in U, the induced subgraph G(V – (U – int(U)) is a disconnected graph with at least two connected components. The interior clique G(int(U)) forms one component while G(V – U) or some subgraph of G(V – U) forms the other. From this observation, we can conclude that the set of vertices defined by

$$\mathrm{ext}(U) = U - \mathrm{int}(U)$$

is a separator of G. This leads us to a property of interior cliques ideally suited for exposing parallelism in sparse symmetric matrices.

Lemma 1. Let *S* be the set of all simplicial vertices in any undirected graph G = (V, E). Then, each connected component of G(*S*) is an interior clique.

For the case where G is a chordal graph, this result was first established by Jess and Kees (1982). The proof that Lemma 1 holds for any undirected graph can be found in Kevorkian (1993).

We now establish a decomposition result that leads to a separator S such that all interior cliques in G = (V, E) are connected components of the induced subgraph G(V – S). The set of vertices S is then used to formulate a four-stage graph-theoretic method for exposing parallelism in general sparse symmetric matrices.

Theorem 1. Let $\Pi = (V_1, V_2, \ldots, V_k)$ be any clique partition in the graph G = (V, E). Then, for any clique G(U) in G, the following relation holds:

$$\mathrm{int}(U) \subseteq \bigcup_{i=1}^{k} \mathrm{cor}(V_i) .$$

Proof. If G(U) is a clique with an empty interior, there is nothing to prove. Suppose G(U) is a clique with a nonempty interior, and let W_Π denote the union of vertex sets $\mathrm{cor}(V_1)$ through $\mathrm{cor}(V_k)$. Assume for contradiction that the assertion in the theorem does not hold. Then we get $\mathrm{int}(U) \cap (V - W_\Pi) \neq \varnothing$, and so for some element V_i of Π we have $\mathrm{int}(U) \cap (V_i - \mathrm{cor}(V_i)) \neq \varnothing$. Let u be any vertex in $\mathrm{int}(U) \cap (V_i - \mathrm{cor}(V_i))$ and let v be any vertex in $\mathrm{cor}(V_i)$. Then we have $v \in V_i$, and so every vertex

in $V_i - \{v\}$ is adjacent to v. This means that vertex $u \in \text{int}(U)$ is adjacent to v since u is in V_i. So we get $V_i \subseteq U$ since any vertex adjacent to a vertex in int(U) must be in U. Now one of the following two cases must hold. Case 1: $v \in \text{int}(U)$. Then we get $\deg_G u = \deg_G v$ since both u and v are in int(U). But since $u \in V_i - \text{cor}(V_i)$ and $v \in \text{cor}(V_i)$, we have $\deg_G u > \deg_G v$ and a contradiction. Case 2: $v \notin \text{int}(U)$. Then $v \in U - \text{int}(U)$ since v is in V_i and $V_i \subseteq U$. Thus we get $\deg_G u < \deg_G v$ since $u \in \text{int}(U)$ and $v \in U - \text{int}(U)$. But since $u \in V_i - \text{cor}(V_i)$ and $v \in \text{cor}(V_i)$, we have $\deg_G u > \deg_G v$ and a contradiction. This completes the proof.

Since each edge of a graph G = (V, E) forms a clique, consider the case where an element V_i of a clique partition Π in G corresponds to an edge (v, w) such that $\deg_G v \neq \deg_G w$. Then, by Theorem 1, we conclude that the vertex with the strictly larger degree in the set $V_i = \{v, w\}$ can never be part of any interior clique in G since $\text{cor}(V_i)$ will exclude this vertex. This simple interpretation of Theorem 1 leads to a separator S such that all interior cliques in G are connected components of induced subgraph G(V – S).

Corollary 1.1. Let S be the set of vertices in G = (V, E) defined by

$$S = \{ v \in V \mid \exists (v, w) \in E \text{ with } \deg_G v > \deg_G w \}.$$

Then every interior clique in G is a connected component of G(V – S).

Proof. Let G(U) be any clique in G with a nonempty interior. Then $\text{int}(U) \cap S$ must be empty if the assertion of the corollary holds. Assume for contradiction that $\text{int}(U) \cap S$ is nonempty, and let v be any vertex in $\text{int}(U) \cap S$. Then there exists an edge (v, w) in E such that $\deg_G v > \deg_G w$ since $v \in S$. Let $U = \{v, w\}$. Then G(U) is a clique in G with $\text{cor}(U) = \{w\}$ since $\deg_G w < \deg_G v$. Now let Π be any clique partition in G with U in Π. Then, by Theorem 1, we have $\text{int}(U) \subseteq W_\Pi$, and so v is in W_Π since $v \in \text{int}(U)$. But this is a contradiction since $v \notin \text{cor}(U)$. Thus $\text{int}(U) \cap S$ is empty, which means that the interior clique G(int(U)) is a subgraph of G(V – S).

Assume for contradiction that G(int(U)) is not a connected component of G(V – S). Then $\text{ext}(U) \cap (V - S)$ must be nonempty since no vertex in int(U) is adjacent to a vertex in V – U. Let v be any vertex in $\text{ext}(U) \cap (V - S)$, and let u be any vertex in int(U). Then there exists an edge (u, v) in E with $\deg_G v > \deg_G u$ since $u \in \text{int}(U)$ and $v \in \text{ext}(U)$. As a result, we have $v \in S$, which is a contradiction since it was assumed that $v \in V - S$. This completes the proof.

<u>Corollary 1.1</u> motivates a four-stage parallelization tool for exposing parallelism in sparse symmetric matrices. The four stages are:

1. Compute the set of vertices S;
2. Compute connected components of induced subgraph G(V – S);
3. Classify clique connected components of G(V – S);
4. Compute independent cliques in nonclique connected components of G(V – S).

Through these four algorithmic stages, we arrive at a vertex partition

$$\Pi^* = (V_1, V_2, \ldots, V_r, S^*),$$

where

$$S \subseteq S^*,$$

satisfying the following three properties:

(a) For any two distinct sets V_i and V_j , no vertex in V_i is adjacent to a vertex in V_j;

(b) For any set V_i , $G(V_i)$ is a clique in G;

(c) The interior of any clique in G is an element of the vertex partition.

By property (a), condensation of G with respect to Π^* is a graph with root vertex S^* and leaf vertices V_1 through V_r .We will refer to the leading r elements of vertex partition Π^* as its leaf elements. Figure 1 illustrates condensation of G with respect to Π^*. The shape of the graph in figure 1, combined with properties (b) and (c), forms the basis for our parallel ordering algorithm.

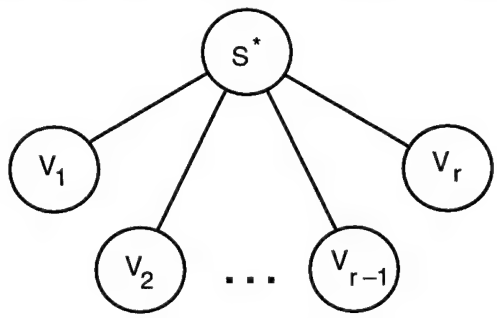
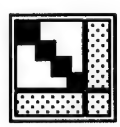


Figure 1. Condensation of G with respect to vertex partition Π^*.

4. A PARALLEL ORDERING ALGORITHM DERIVED FROM VERTEX PARTITION Π^*

Let M be any symmetric matrix with undirected graph G = (V, E). By property (a) of vertex partition Π^*, there exists in M an r-by-r block diagonal matrix A with diagonal blocks A_1 through A_r such that block A_i corresponds to induced subgraph $G(V_i)$ for $1 \le i \le r$. Thus, by property (a) of vertex partition Π^*, there exists a permutation matrix P such that PMP^T has the block bordered diagonal form

$$PMP^T = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix}. \tag{1}$$



Property (b) of vertex partition Π^* ensures that each diagonal block of the leading block A is a full matrix (we assume that the original matrix M has a nonzero main diagonal). We will refer to A as the block pivot in PMP^T.

Without loss of generality, suppose the block pivot A is nonsingular, and let U be the upper Cholesky factor of A. Then the block matrix PMP^T can be written in the block product form

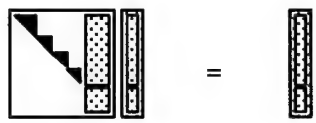
$$PMP^T = \begin{bmatrix} U^T & 0 \\ X^T & I \end{bmatrix} \begin{bmatrix} U & X \\ 0 & D-X^TX \end{bmatrix} \tag{2}$$

in which I is an identity matrix, the 0's are zero matrices, and the block X is the solution of the multiple right-hand side triangular system $U^TX = B$. The matrix $D - X^TX$ is the familiar Schur complement of A in M. Replacing M by PMP^T in the original system of equations $Mx = b$, we obtain the following equivalent system:

$$(PMP^T)(Px) = (Pb). \tag{3}$$

Consider this system where PMP^T has been factored into the block product form (equation 2) and the vectors Px and Pb have been partitioned conformably into the direct sums of y and z, and f and h, respectively. The system of equations (equation 3) then becomes

$$\begin{bmatrix} U & X \\ 0 & D-X^TX \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} w \\ h-X^Tw \end{bmatrix} \tag{4}$$



where w is the solution of the lower triangular system $U^Tw = f$. The zero-nonzero structure of the Cholesky factor U in equation 4 serves to illustrate the point that all r diagonal blocks of A can be factored in parallel, and that block X can be computed from $U^TX = B$ by solving r triangular systems with multiple right-hand sides in parallel.

If the Schur complement $D - X^TX$ in equation 4 is dense, the solution of $Mx = b$ is obtained by solving the two systems of equations $(D - X^TX)z = h - X^Tw$ and $Uy = w - Xz$ in that order, and setting $x = P^T[y\ z]^T$. However, in many industrial sparse matrix applications, the Schur complement $D - X^TX$ is generally sparse and large. Therefore, to deal with the most general case, we need an algorithm that recursively uses the four-stage parallelization tool until the symbolic form of the Schur complement $D - X^TX$ is either full or dense. In the Algol-like language adopted by Aho, Hopcroft, and Ullman (1976), the new sparse matrix ordering algorithm takes the following form:

```
procedure parallel_ordering:
while G is not a clique do
    begin
        search;
        if S is not empty then
            dfs
        else
            cliques(V)
        end;
        comment we now have vertex partition Π* or block pivot A in PMP^T;
        sfp;
        comment procedure sfp computes symbolic form Δ of D – X^TX;
        replace G = (V, E) by graph of Δ
    end
```

Procedure search computes the set of vertices S. If S is empty, we call procedure cliques(V) to compute independent cliques in G. If S is not empty, we call procedure dfs (depth-first search) to compute the connected components of induced subgraph G(V – S). For each connected component G(U) of G(V – S), dfs invokes a procedure called classify(U) to analyze G(U). If G(U) is a clique, then the set of vertices U is a leaf element of vertex partition Π^*. If G(U) is not a clique, then

procedure classify(U) calls cliques(U) to compute independent cliques in G(U). For each independent clique $G(U')$ in G(U), the set of vertices U' is a leaf element of vertex partition Π^*. At the completion of procedure dfs, the computation of vertex partition Π^* or block pivot A in PMP^T is complete. Subsequently, we call procedure sfp (symbolic factorization procedure) to compute the symbolic form Δ of the Schur complement $D - X^TX$. The symbolic form of block X in equation 2 is computed in procedure classify. Finally, we set G to the graph of the symbolic Schur complement and repeat the above steps until G is a clique. In practice, the recursion in the parallel ordering algorithm may be terminated when the Schur complement becomes very dense.

5. IMPLEMENTATION OF THE PARALLEL ORDERING ALGORITHM

In this section, we give detailed descriptions of five procedures used in the parallel ordering algorithm.

5.1 COMPUTING THE SET OF VERTICES S

We use procedure search to compute the vertex set S in linear time by visiting the vertices and edges of a graph G = (V, E) as follows. We select and visit a vertex v. Then, for each vertex w adjacent to v, we do the following. If w has been visited previously, we pick another vertex adjacent to v. If w has not been visited previously, we compare the degrees of vertices v and w. If $\deg_G v = \deg_G w$, we pick another vertex adjacent to v. If $\deg_G v \neq \deg_G w$, then the vertex with the strictly larger degree is marked as the vertex belonging to the set S. This process is continued until all vertices in V have been visited. At the completion of procedure search, all vertices in S are marked "old" and all vertices in V – S are marked "new." An implementation of procedure search can be found in Kevorkian (1993).

We conclude this subsection with another key property of the vertex set S that shows that the induced subgraph G(V – S) not only contains all interior cliques of G but also all cliques whose vertices have minimum degree in G.

Lemma 2. Let G(U) be any clique in G = (V, E) such that $\deg_G u = \delta(V)$ for all $u \in U$. Then, G(U) is a clique in G(V – S).

Proof. Let u be any vertex in U. Then $\deg_G u = \delta(V)$, and so for any edge (u, v) incident with u we have $\deg_G u \leq \deg_G v$. Thus $u \notin S$, and the proof is complete.

5.2 COMPUTING CONNECTED COMPONENTS OF G(V – S)

Procedure dfs uses the linear-time depth-first search method (Aho, Hopcraft, & Ullman, 1976; Tarjan, 1972) to compute all connected components of induced subgraph G(V – S). For each vertex v in V – S marked "new," dfs calls a recursive procedure component(v) to compute a connected component G(U) of G(V – S). The entire algorithm is given below. Since dfs is called immediately after search, all vertices in V – S are marked "new," and all vertices in S are marked "old" at start of dfs.

```
procedure dfs:
for all v in V do
    if v is marked "new" then
        begin
            mark v "old";
            U ← {v};
            REU ← 0;
            NGU ← empty;
            component(v);
            classify(U)
        end
procedure component(v):
for each vertex w adjacent to v do
    if w is in V – S  then
        begin
            REU ← REU + 1;
            if w is marked "new" then
                begin
                    mark w "old";
                    add w to U;
                    component(w)
                end
        end
else
    if w is not on NGU then add w to NGU
```

The integer REU keeps count of each edge whose two end points are in the set U, and so on completion of connected component G(U) we get REU = 2 × |E(U)| since the depth-first search method visits each edge of G exactly twice (Aho, Hopcraft, & Ullman, 1976). Thus, a connected component G(U) of G(V – S) is a clique if and only if the integer REU satisfies the following equality

$$REU = |U| \times (|U| - 1) \tag{5}$$

since a clique with |U| vertices has |U| × (|U| – 1)/2 edges. This simple algebraic relation will be used later to categorize all connected components of G(V – S) into cliques and noncliques.

While computing the connected component G(U), dfs also computes a set of vertices $N_G U$ defined by

$$N_G\, U = \{w \in V - U \mid w \text{ is adjacent to a vertex in } U\}.$$

We will refer to the set of vertices $N_G U$ as the neighborhood of G(U) in G. The single array NGU computed in component(v) stores the neighborhood of each connected component of G(V – S) one at a time.

5.3 CLASSIFYING THE CLIQUES IN G(V – S)

Upon completion of connected component G(U) in dfs, procedure classify(U) is called for further analysis of G(U). If G(U) is a clique, then U is a leaf element of vertex partition Π^*. If G(U) is not a clique, procedure cliques(U) is called in classify(U) to compute independent cliques in G(U). For

each independent clique $G(U')$ in $G(U)$, U' is a leaf element of vertex partition Π^*. Each leaf element of every vertex partition computed in parallel_ordering is placed on a single array VP in the order they get computed. At completion of parallel_ordering, the array VP together with the end element S^* of the final vertex partition provide the ordering generated by parallel_ordering.

For each connected component $G(U)$ of $G(V - S)$ that is a clique, procedure classify(U) categorizes $G(U)$ so that the symbolic factorization of the block pivot A is facilitated. The basic methodology involved in the classification is covered next.

Let $G(U)$ be any clique in the induced subgraph $G(V - S)$. Then the following two conditions are of interest in the symbolic factorization of block pivot A in PMP^T.

(a) For any $u \in U$, $adj_G u \cap S = N_G U$,

(b) Subgraph induced by $N_G U$ is a clique.

By conditions (a) and (b), every connected component $G(U)$ of $G(V - S)$ that is a clique must be one of four distinct types. These are:

Type C_1: $G(U)$ satisfies (a) and (b);
Type C_2: $G(U)$ satisfies (a) and not (b);
Type C_3: $G(U)$ satisfies (b) and not (a);
Type C_4: $G(U)$ satisfies neither (a) nor (b).

The next series of results provides properties we use in computing the symbolic forms of block X and the Schur complement $D - X^T X$.

Lemma 3. For any connected component $G(U)$ of $G(V - S)$, $G(U)$ is an interior clique if and only if $G(U)$ is a type C_1 clique.

Proof. Suppose $G(U)$ is an interior clique. Then there exists in G a clique $G(C)$ with $U \subseteq C$ such that any vertex u in U is adjacent to all other vertices in C. Thus we get $adj_G u = C - \{u\}$, which gives us $adj_G u = (N_G U \cup U) - \{u\}$ since C consists of the two disjunct sets U and $N_G U$. But since clique $G(U)$ is a connected component of $G(V - S)$, we have $adj_G u \cap (V-S) = U - \{u\}$. Therefore, we get $adj_G u \cap S = N_G U$, which is condition (a). Condition (b) holds since $G(C)$ is a clique and $N_G C \subset C$. Hence, any interior clique in G is a type C_1 clique.

Suppose $G(U)$ is a type C_1 clique. Let $C = U \cup N_G U$. Then, by conditions (a) and (b), $G(C)$ is a clique. Also, by condition (a), we have $deg_G u = |U| - 1$ for any $u \in U$. Thus, we get $U = int(C)$, and so $G(U)$ is an interior clique. This completes the proof.

In view of Lemma 3, we call type C_1 and type C_2 cliques semi-interior, while a strictly type C_2 clique is called strictly semi-interior.

Lemma 4. For any connected component $G(U)$ of $G(V - S)$, $G(U)$ is a semi-interior clique if and only if the following equality holds:

$$deg_G\, v = |N_G U| + |U| - 1, \qquad \text{for any } v \in U. \quad (6)$$

Proof. If clique $G(U)$ is semi-interior, condition (a) holds, and so for any $v \in U$ we get $|adj_G u \cap S| = |N_G U|$. Also, we have $|adj_G u \cap (V - S)| = |U| - 1$ since $G(U)$ is a clique and a connected component of $G(V - S)$. Combining these two equalites gives relation 6. Suppose relation 6 holds. Then, for any $u \in U$, we have $|adj_G u| = |N_G U| + |U - \{u\}|$. But since $G(U)$ is a connected component of $G(V - S)$,

no subset of U is in S, and so we obtain $|adj_G u \cap S| = |N_G U|$. Hence, condition (a) holds, which means that G(U) is a semi-interior clique. This completes the proof.

Lemma 5. Let G(U) be any semi-interior clique in G(V – S). Then, for any v in U, the following two statements hold.

(i) $def_G\, v = \{ (u, w) \mid u, w \in N_G U, (u, w) \notin E, u \neq w \}$.

(ii) The graph $(N_G U, E(N_G U) \cup def_G\, v)$ is a clique.

Proof. Let u and w be any two distinct vertices in $adj_G v$. Then each of the vertices u and w is either in U or $N_G U$. If both u and w are in U, then we have $(u, w) \in E$ since G(U) is a clique. If u is in U and w is in $N_G U$, then by condition (a) we get $(u, w) \in E$. Thus, the pair (u, w) can never be an edge in $def_G u$ if either u or w is in U. Hence, both u and w must be in $N_G U$ if (u, w) is an edge in $def_G v$. This completes the proof for statement (i). Combining statement (i) and condition (a) yields statement (ii). This completes the proof.

Procedure classify(U) employs Lemma 4 and Lemma 5 to simplify the symbolic factorization of block pivot A in PMP^T. To highlight this, let us first assume that G(U) is a semi-interior clique. Also, let G(U) be the graph of the ith diagonal block A_i in block pivot A. Then, by statement (i) of Lemma 5, the symbolic factorization of A_i does not produce any fill-in in block X since both end points of any edge in $def_G v$ represent two distinct rows of the Schur complement $D - X^T X$. Furthermore, by statement (ii) of Lemma 5, the symbolic factorization of any single row of block A_i generates all the fill-ins produced by the symbolic factorization of the entire block A_i. Therefore, to compute the fill-ins generated by the symbolic factorization of block A_i, we can choose any vertex u in U and ignore all other vertices in U – {u} and all edges incident with each of the vertices in U – {u}.

Suppose the clique G(U) is not semi-interior. Then the symbolic factorization of block A_i may produce fill-ins in block X. To compute the symbolic form of block X, classify(U) invokes a procedure called xfills(U). The entire procedure xfills is given below. We use a Boolean array TEST setting TEST(x) = false at the start of procedure xfills if and only if vertex x is in V – U. Also, we use the Matlab colon notation (The Mathworks, 1990) to let U = U(1:|U|) and let U(i) designate the ith element of U.

```
procedure xfills(U):
for i ← 1 until |U| – 1 do
    for each vertex x adjacent to U(i) do
        if TEST(x) is false then
            begin
                TEST(x) ← true;
                for each vertex y on U(i + 1: |U|) do
                    add x to the adjacency of y;
                add end vertex on U to adjacency of x
            end
        end
```

The general step in procedure xfills is as follows. Let u = U(i) and let y be any vertex on U(i+1: |U|) for i = 1, ... , |U| – 1. Then y is adjacent to u since G(U) is a clique, and so for any vertex x in V – U adjacent to u we have $(x, y) \in def_G u \cup E$. Thus, for each vertex y on U(i+1:|U|), (x, y) is either an edge in E or a vertex pair corresponding to a fill-in in block X. SupposeTEST(x) = true at the start of the ith iteration of the main **for** loop. Then, by the initialization of array TEST, vertex x must be adjacent to a vertex in U(1: i – 1), and so the pair (x, y) has already been accounted for in a previous

iteration of the main **for** loop. Suppose TEST(x) = false. Then the following three steps are carried out. First, TEST(x) is set true so that the pair (x, y) is not considered further. Second, the symbolic form of X is updated by adding x to the adjacency of each vertex on U(i + 1: |U|). Third, among all the vertices on U(i + 1: |U|), we only add the end vertex on U to the adjacency of x. This step is done to facilitate the application of procedure sfp. It is important to note that the updates of the adjacencies of vertices x and y in xfills may give rise to duplicate edges. However, in practice we use a more detailed version of xfills that avoids duplicate edges without affecting its time complexity.

At the completion of procedure xfills(U), the end vertex on U is adjacent to all vertices in $N_G U$. Thus, the end vertex on U satisfies statement (ii) of Lemma 5 at the completion of xfills, and so the symbolic factorization of the single row of block A_i corresponding to the end vertex on U will generate all the fill-ins in $D - X^T X$ produced by the factorization of the entire block A_i. In subsequent developments, we call a vertex u in any clique G(U) a "vertex of choice" if the symbolic factorization of the single row of A_i corresponding to u generates all the fill-ins in $D - X^T X$ produced by the symbolic factorization of the entire block A_i.

The entire procedure classify(U) is given below. We use a Boolean array VOC, setting VOC(y) to true if and only if y is a vertex of choice. If G(U) is not a clique, then cliques(U) is called to compute independent cliques in G(U). Suppose G(U) is a clique. Then the vertex set U is a leaf element of the vertex partition Π^*, and so U is placed on array VP. If clique G(U) is not semi-interior, procedure xfills(U) is called to compute the symbolic form of block X. Subsequently, the end vertex on U becomes a vertex of choice in G(U). If clique G(U) is semi-interior, then procedure xfills(U) is skipped since no fill-ins are produced in block X. Next, the end vertex on U is chosen as vertex of choice. We assume that the array VP is empty and Boolean array VOC is set false at the start of the parallel ordering algorithm.

```
procedure classify(U):
if REU = |U| × (|U| – 1) then
    begin
        comment G(U) is a clique;
        add U to VP;
        if deg_G v ≠ |NGU| + |U| – 1 then xfills(U);
        VOC(U(|U|)) ← true ;
        comment end vertex on U is vertex of choice
    end
else
    cliques(U)
end
```

The concept of a semi-interior clique is closely related to a notion widely used in the literature. To establish this connection, let G(U) be any semi-interior clique in G(V – S). Then, by condition (a), any two vertices x and y in the set U satisfy the equality

$$\{ x \} \cup adj_G x = \{ y \} \cup adj_G y.$$

Vertices satisfying this equality are called "indistinguishable" vertices in G (George & Liu, 1989). Thus, for any semi-interior clique G(U) in G, all vertices of G(U) are indistinguishable. If clique G(U) is not semi-interior, then there must exist in U at least two distinct vertices x and y that are not indistinguishable.

An important feature of indistinguishable vertices relates to the merging of these vertices together as one vertex called a supernode (George & Liu, 1981, 1989). This way, a supernode becomes a

"representative" (George & Liu, 1989) in the group of indistiguishable vertices. In our parallel ordering algorithm, a vertex of choice plays the same role as a supernode does, although no merging of vertices is required in our formulation. Also, in the case where all the vertices in a clique G(U) are not indistinguishable, the approach we have followed to do symbolic factorization allows us to transform a vertex in a nonsemi-interior clique into a vertex of choice.

5.4 COMPUTING INDEPENDENT CLIQUES IN G(V – S)

For any nonclique connected component G(U) of G(V – S), we use procedure cliques(U) to compute independent cliques in G(U). For each independent clique G(W) computed in cliques, we place W on array VP and call procedure xfills(W) to compute the fill-ins in block X. The entire procedure is given below. At the completion of the call to xfills(W), the end vertex on W becomes a vertex of choice. We assume all vertices in U are marked "new" at the invocation of cliques(U).

```
procedure cliques(U):
while there is a vertex in U marked "new" do
    begin
        compute a maximal clique G(W) in G(U) with all u ∈ W marked "new";
        add W to VP;
        xfills(W);
        mark all vertices in W and N_G W "old";
        add N_G W to S;
        VOC(U(|W|)) ← true
    end
```

Let G(W) be any maximal clique computed in cliques(U). Since all vertices in W and $N_G W$ are marked "old" at the completion of each iteration of the **while** loop, no vertex of any maximal clique computed thereafter in procedure cliques(U) is adjacent to a vertex of G(W). Thus, all maximal cliques computed at the completion of procedure cliques(U) are independent. All vertices in G(U) that are not in any independent clique computed in cliques(U) are added to the set of vertices S constructed earlier in procedure search. Thus, if S' denotes the set of all vertices added to the set S in cliques(U) at the completion of the four-stage parallelization tool, then we have $S \cup S' = S^*$, where S^* is the end element of vertex partition Π^*.

5.5 COMPUTING THE SYMBOLIC SCHUR COMPLEMENT

Let G = (V, E) be the undirected graph of any sparse symmetric matrix M and let

$$\Pi^* = (V_1, V_2, \ldots, V_r, S^*)$$

be the vertex partition obtained by applying algorithm parallel_ordering to matrix M. Then, by the construction of the block bordered diagonal matrix PMP^T in equation 1, each vertex in the set S^* represents a row of the square diagonal block D in PMP^T. Thus, assuming no lucky cancellation of nonzero elements, the undirected graph of the Schur complement $D - X^TX$ consists of the pair $G^* = (S^*, E^*)$ where $E(S^*)$ corresponds to the nonzero off-diagonal entries in block D, while each edge in the set $E^* - E(S^*)$ corresponds to a fill-in produced in $D - X^TX$.

The first graph-theoretic characterization of Cholesky factorization is due to Parter (1961). Subsequently, George and Liu (1981) gave a graph-theoretic charcterization of block Cholesky factorization using the notion of reachable sets. Using the block product form of PMP^T in equation 2, the reachable set for any vertex v in the set S^* in G can be defined as the set of vertices given by

$$\text{Reach}(v, V-S^*) = \{w \in S^* \mid \exists \text{ a path from } v \text{ to } w \text{ containing only } v, w, \text{ and vertices in } V-S^*\}.$$

For the graph $G^* = (S^*, E^*)$ of the Schur complement $D - X^TX$, the main property of the reachable set states (George & Liu, 1981) that

$$\text{adj}_{G^*}\, v = \text{Reach}(v, V - S^*) \qquad \text{for any } v \text{ in } S^*. \quad (7)$$

In other words, for any vertex w in Reach(v, V – S*), the pair (v, w) is either an edge of the original graph G or an edge corresponding to a fill-in in the Schur complement $D - X^TX$. It is easy to see that George and Liu's relation (7) is a generalization of Parter's original characterization of Cholesky factorization (Parter, 1961).

As pointed out in George and Liu (1981), the work required to compute reachable sets can be excessively large, especially for the case where $|V - S^*|$ is large. To overcome this drawback, George and Liu (1981) introduce the quotient graph model, where supernodes are used in the process of elimination. Using this strategy, the work done in computing reachable sets is reduced dramatically by limiting the lengths of paths required to compute reachable sets to one or two.

In the rest of this section, we give another simple result that makes use of the vertices of choice to characterize the adjacency of each vertex of the graph of the Schur complement $D - X^TX$ in terms of the adjacencies in the original graph G = (V, E).

For any vertex v in the set of vertices S^*, let $\Psi(v)$ be the set of vertices in G defined by

$$\Psi(v) = \{\, u \in V - S^* \mid u \in \text{adj}_G v \text{ and } \text{VOC}(u) = \text{true}\}.$$

Thus, for any vertex v in S^*, $\Psi(v)$ comprises all vertices of choice that are adjacent to v in G.

Lemma 6. Let $G^* = (S^*, E^*)$ be the graph of the Schur complement $D - X^TX$. Then, for any vertex v in S^*, the following relation holds.

$$\text{adj}_{G^*} v = (\text{adj}_G v) \cup \Big(\bigcup_{u \in \Psi(v)} (\text{adj}_G u \cap S^*) - \{v\}\Big).$$

Proof. Let (v, w) be any edge in E^*. If (v, w) is in E, then w is adjacent to v in G, and so the lemma holds since $\text{adj}_G v \subseteq \text{adj}_{G^*} v$. Suppose (v, w) is not in E. Then there is a path from v to w containing only v, w, and vertices in $V - S^*$. Let x and y be the vertices in $V - S^*$ adjacent to v and w, respectively. Since each connected component of $G(V - S^*)$ is a clique, there exists in $G(V - S^*)$ a clique G(U) containing vertices x and y. By construction of procedures classify and cliques, each connected component of $G(V - S^*)$ contains a vertex of choice. Let u be the vertex of choice in G(U). Then both v and w are adjacent to u since v and w are in the neighborhood $N_G U$ of U. Thus, u is in $\Psi(v)$ and w is in $\text{adj}_G u$, which means that w is in $\text{adj}_{G^*} v$. This completes the proof.

We present an implementation of procedure sfp that uses Lemma 6 to compute the symbolic form of the Schur complement in time proportional to $|V| + |E| + |E^* - E(S^*)|$. The entire procedure sfp is given below. The vertex set S at the start of procedure sfp is the same as the set S^* in Lemma 6. We use the Boolean array TEST in sfp to avoid duplicates of the edges corresponding to the fill-ins in $D - X^TX$. We assume that array TEST is set to false at the start of procedure sfp.

```
procedure sfp:
for each vertex v in S do
    begin
        set Ψ(v) to empty;
        for each vertex u adjacent to v do
            if u is in S then TEST(u) ← true;
            else
                begin
                    delete u from adjacency of v;
                    if VOC(u) = true then add u to Ψ(v)
                end;
            comment computation of Ψ(v) is complete;
            reach(v)
    end

procedure reach(v):
begin
    TEST(v) ← true;
    U ← [v];
    for each vertex u on Ψ(v) do
        for each vertex x adjacent to u do
            if x is in S then
                if TEST(x) = false then
                    begin
                        comment (v, x) corresponds to a fill-in;
                        add x to adjacency of v;
                        add v to adjacency of x;
                        TEST(x) ← true;
                        add x to U
                    end;
            else
                delete x from adjacency of u;
    for each x on U do TEST(x) ← false
end
```

For each vertex v in S, procedure sfp carries out the following general step to compute the set $\Psi(v)$. Let u be any vertex adjacent to v. If u is in S, then u is not a vertex of choice since all vertices of choice are in V – S. Consequently, we use the Boolean array TEST to mark u true so that the edge (v, u) in E is not mistaken for an edge in $E^* - E(S^*)$. Suppose u is not in S. Then we delete u from the adjacency of v since the vertices in V – S are not required in subsequent iterations of the **while** loop in the new ordering algorithm. Subsequently, we test if u is a vertex of choice. If the test holds, we add u to the set $\Psi(v)$. The application of this step to all vertices adjacent to v completes the computation of the set $\Psi(v)$.

On the completion of set $\Psi(v)$, we call procedure reach(v) to compute the fill-ins produced in $D - X^TX$ by the factorization of the row of M corresponding to vertex v. The general step in procedure reach(v) is as follows. First, we mark TEST(v) true to exclude vertex v from the set of vertices $adj_{G^*}v$ defined in Lemma 6. Next, we initialize the set U to [v]. The main purpose of the set U is to help reset the Boolean array TEST to false at the completion of reach(v). For each vertex u on $\Psi(v)$, we visit every vertex x adjacent to u. If vertex x is not in S, then the pair (v, x) cannot be an edge

of G^*. At this point, we complete the visit at vertex x by deleting x from the adjacency of u so that at the next visit to vertex of choice u in reach(v), the edge (u, x) is not visited again. Suppose x is in the set S. Then we proceed with the evaluation of the condition TEST(x) = false. If TEST(x) = false, then by Lemma 6 the edge (v, x) corresponds to a fill-in in the Schur complement $D - X^TX$. As a result, we add x to the adjacency of v, and v to the adjacency of x. Also, we set TEST(x) = true to avoid duplicate edges in the edge set E^*. If TEST(x) $\neq$ false, then (v, x) is an edge in E^* and so we visit the next vertex of choice on $\Psi(v)$. After all vertices of choice on $\Psi(v)$ have been visited, TEST is reset to false using array U. Subsequently, reach(v) returns to procedure sfp. This completes the computation of the symbolic form of the Schur complement.

We complete our analysis by showing that procedure sfp requires time proportional to $|V| + |E| + |E^* - E(S^*)|$. For each vertex v in $S \subseteq V$, sfp constructs $\Psi(v)$ by visiting each edge incident with v exactly once. Thus the total work done in sfp, excluding all calls to procedure reach, is proportional to $|V| + |E|$. Let u be any vertex on $\Psi(v)$, and let x be any vertex in V adjacent to u. If x is not in S, then the edge (u, x) is visited once in sfp since x is deleted from adjacency of u in reach(v). Suppose the vertex x is in S. Then TEST(x) is either true or false. If TEST(x) = true, then (v, x) is an edge in $E(S^*)$. If TEST(x) = false, then (v, x) is an edge in $E^* - E(S^*)$, or equivalently, (v, x) is an edge corresponding to a fill-in in the Schur complement $D - X^TX$. Thus for each edge (v, u) incident with a vertex v in S and a vertex of choice u, procedure reach traverses a path (v, u, x) of length 2 where (v, x) is an edge either in $E(S^*) \subseteq E$ or in $E^* - E(S^*)$. Hence, the total work done in sfp is proportional to $|V| + |E| + |E^* - E(S^*)|$.

6. COMPUTATIONAL RESULTS, COMPARISONS, AND COMPLEXITY

In this section, we evaluate the new parallel ordering algorithm and compare it with two commonly used ordering methods implemented in Matlab (The Mathworks, 1990). These are the minimum degree method and the reverse Cuthill-McKee band reduction method (George & Liu, 1981). The Matlab implementation of the minimum degree ordering incorporates many of the latest developments of minimum degree as well as ideas for enhancing parallel sparse matrix factorizations (Gilbert, Molor, & Schreiber, 1992). The reverse Cuthill-McKee was included in the evaluation since the benchmark contained banded matrices. The benchmark for the comparisons comprised a set of 21 sparse symmetric matrices drawn from applications of linear and quadratic programming to industrial problems (Gill et al., 1991, 1992; Saunders, private communication, 1992–1993) and from the Harwell–Boeing sparse matrix collection (Duff, Grimes, & Lewis, 1989).

Table 1 summarizes the results for the sparse matrices from linear and quadratic programming applications. The heading Identifier/n/nz in the first column of table 1 refers to the name of the specific application, the order of the sparse symmetric matrix M, and the number of nonzeros in M, respectively. The second column (Method) includes the Matlab minimum degree method (mmd), the Matlab Reverse Cuthill-McKee method (rcm), and the new parallel ordering algorithm (pal). The third column (Nz) gives the total number of nonzero entries in $R^T + R$, where R is the upper Cholesky factor of PMP^T. The fourth column (Fill-In) gives the number of fill-ins created in the upper Cholesky factor R by each of the three methods (rcm, mmd, and pal). The fifth and last column gives the elimination tree height for each of the three methods.

For the second through tenth problems in table 1, the new ordering algorithm performs better than the minimum degree and Reverse-Cuthill methods in the number of fill-ins. It also performs better than the minimum degree method in computing elimination trees with smaller heights. It is worth noting that for problem scfxm1, the height of the elimination tree of the Cholesky factor computed

by the new ordering method is about 65 percent of the elimination tree height computed by the minimum degree method. Figures 2 and 3 illustrate the zero-nonzero structures of $R^T + R$ for scfxm1 and shell.

Table 1. Examples from linear and quadratic programming applications.

Identifier/N/Nz	Method	Nz (R^T+R)	Fill-In (R)	Elimination Tree Height
share1b/117/1885	rcm	3360	796	71
	mmd	2538	385	43
	pal	2737	426	46
israel/214/5682	rcm	25574	9946	187
	mmd	7234	776	60
	pal	7168	743	57
brandy/255/5273	rcm	18767	6747	160
	mmd	8893	1810	80
	pal	8835	1781	78
bandm/320/6848	rcm	21134	7143	242
	mmd	10848	2000	73
	pal	9180	1166	65
scfxm1/473/5613	rcm	22463	8425	306
	mmd	15083	4735	105
	pal	11409	2898	68
seba/529/7639	rcm	69157	30759	281
	mmd	9003	682	43
	pal	8851	606	32
fffff800/616/16446	rcm	160038	71796	482
	mmd	35638	9596	137
	pal	34922	9238	132
shell/651/3809	rcm	19435	7813	234
	mmd	8925	2558	75
	pal	8519	2355	58
pilot/1129/10733	rcm	151817	70522	751
	mmd	43181	16204	152
	pal	37451	13339	142
25fv47/1301/22307	rcm	253359	115526	807
	mmd	102797	40245	265
	pal	83501	30597	232

Table 2 summarizes the results for 11 problems taken from the Harwell–Boeing sparse matrix collection. These problems were drawn from five distinct application areas. These are reservoir modeling (pores_2 and pores_3); finite element approximations to structural engineering problems (nos1 and nos5); aircraft design problems (can_292, can_1054, and can_1072); structural engineering matrices from the aerospace industry (bcsstk19 and bcsstk20); and power systems admittance matrices (685_bus and 1138_bus). The fill-ins produced in the Cholesky factor R of PMP^T were validated by applying the Matlab symbolic factorization utility "symbfact" to M(P, P) for each matrix M considered in tables 1 and 2.

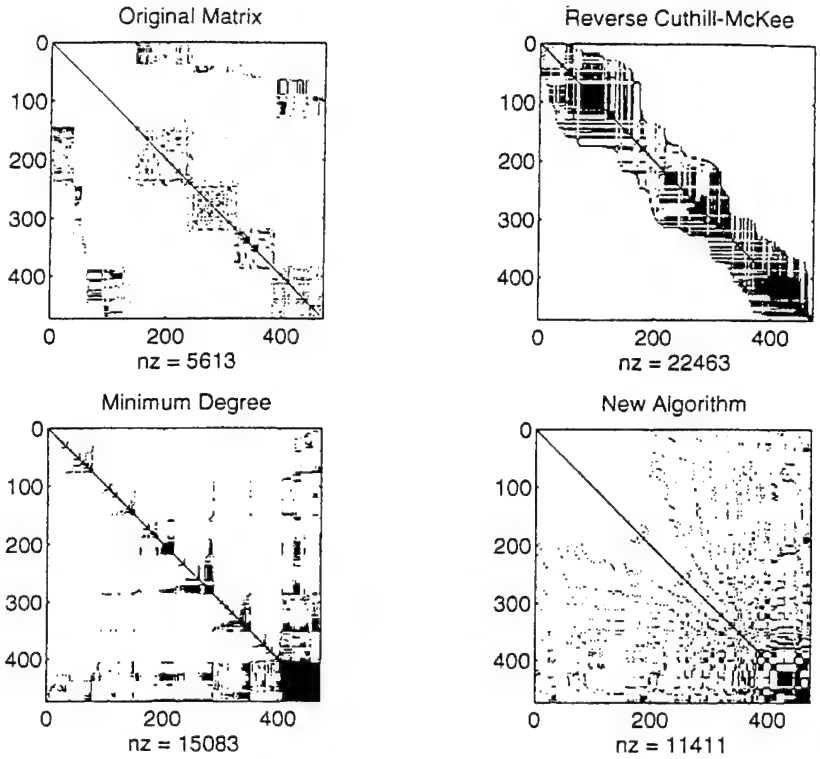


Figure 2. Sparsity structure of R^T+R for problem scfxm1.

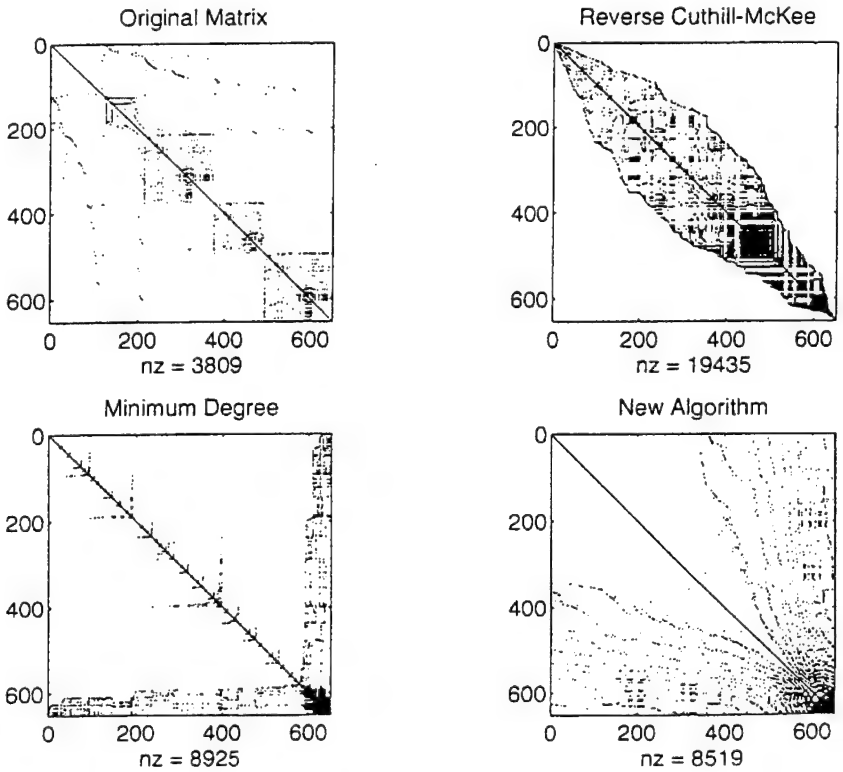


Figure 3. Sparsity structure of R^T+R for problem shell.

Among the three methods considered in table 2, the parallel ordering algorithm pal computes Cholesky factors with smaller elimination tree heights for 10 problems. For problems nos1 and bcsstk20, the elimination tree heights computed by the parallel ordering algorithm are more than 62 percent smaller than the elimination tree heights computed by the next best method, which is mmd. With regard to the number of fill-ins, the parallel ordering algorithm performs better for six problems, the minimum degree method for three problems, and the Reverse Cuthill–McKee for two problems. Figures 4 and 5 illustrate the zero–nonzero structures of $R^T + R$ for pores_3 and 685_bus.

Table 2. Examples from Harwell–Boeing sparse matrix collection.

Identifier/N/Nz	Method	Nz (R^T+R)	Fill-In (R)	Elimination Tree Height
pores_3/532/3054	rcm	7648	2292	435
	mmd	6908	1922	57
	pal	6506	1721	49
pores_2/1224/9992	rcm	110954	50481	1056
	mmd	53366	21687	273
	pal	51822	20915	183
nos1/237/1017	rcm	1117	77	156
	mmd	1577	280	43
	pal	1941	462	16
nos5/468/5172	rcm	46608	20718	443
	mmd	36028	15428	198
	pal	36618	15723	147
can_292/292/2540	rcm	12988	5244	201
	mmd	5590	1525	43
	pal	5088	1274	40
can_1054/1054/12196	rcm	76904	32354	1001
	mmd	47786	17795	162
	pal	44746	16275	173
can_1072/1072/12444	rcm	97673	42617	956
	mmd	46464	17010	161
	pal	45600	16578	147
bcsstk20/485/3135	rcm	4801	833	260
	mmd	5107	986	83
	pal	5209	1037	30
bcsstk19/817/6853	rcm	18659	5903	762
	mmd	17047	5097	139
	pal	22181	7663	93
685_bus/685/3249	rcm	17977	7634	332
	mmd	7229	1990	57
	pal	6935	1843	51
1138_bus/1138/4054	rcm	9312	2629	229
	mmd	5640	793	35
	pal	5686	816	29

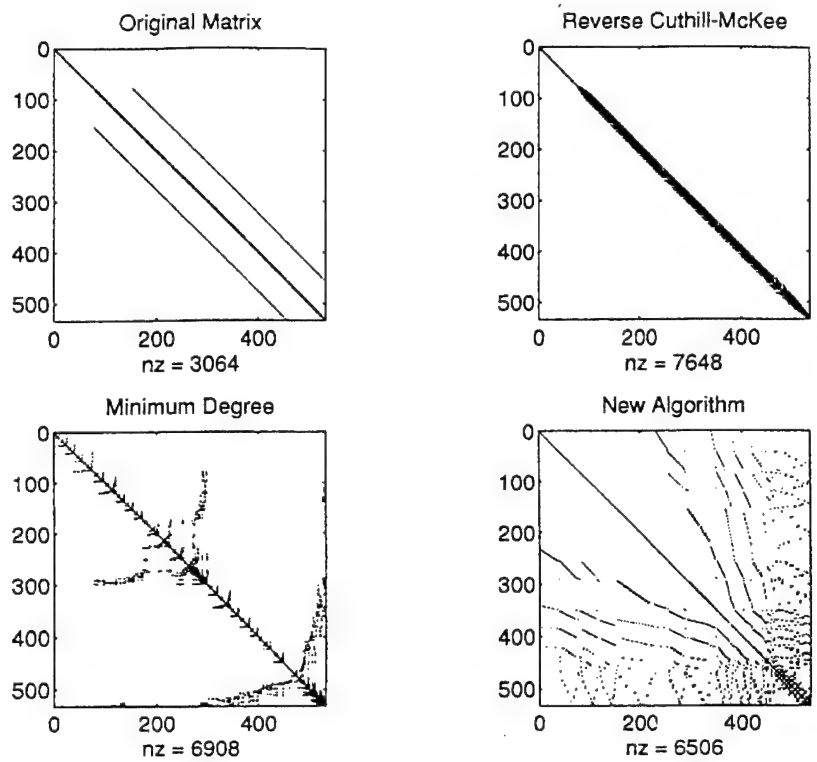


Figure 4. Sparsity structure of R^T+R for problem pores_3.

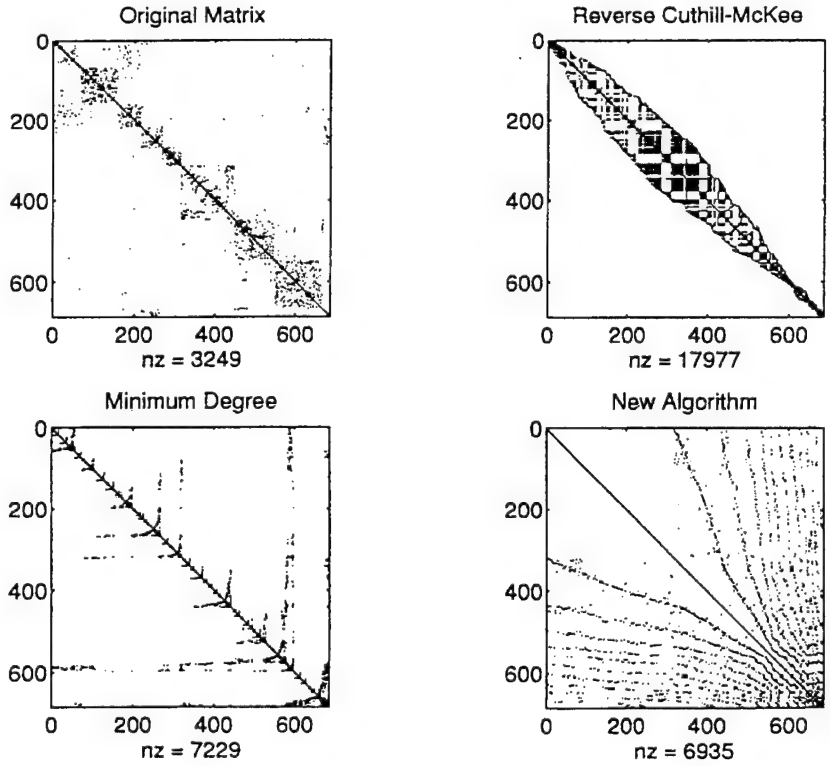


Figure 5. Sparsity structure of R^T+R for problem 685_bus.

Problem nos1 in table 2 is an interesting case since it captures the best and worst performances of the parallel ordering algorithm pal relative to the minimum degree method. This problem comprises a banded matrix (Duff, Grimes, & Lewis, 1989) with very small bandwidth, and so we can explain the performance of pal by considering an n-by-n tridiagonal matrix M with $n \geq 6$. Let $G = (V, E)$ be the undirected graph of M. Then, at the completion of procedure search, we have $S = \{v_2, v_{n-1}\}$, and so the induced subgraph $G(V - S)$ at the completion of search consists of three connected components, which are $G(\{v_1\})$, $G(\{v_n\})$, and $G(U)$, where $U = \{v_3, \ldots, v_{n-2}\}$. Since $n \geq 6$, $G(U)$ is a path with length > 1, and so procedure cliques(U) is called to compute independent cliques in G(U). Assuming the vertices $v_3, \ldots, v_{n-2}$ in U are visited in that order, the first independent clique computed in G(U) is G(W), where $W = \{v_3, v_4\}$. Procedure cliques(U) marks v_4 as the vertex of choice in G(W) since v_4 is the last vertex visited in the process of computing G(W). Also, vertex v_5 is added to the set S since v_5 is adjacent to a vertex in W. Now, since vertex v_3 in W is adjacent to vertex v_2 in S and $(v_4, v_2) \notin E$, the call to xfills(W) in cliques(U) produces the fill-in corresponding to the pair (v_4, v_2). Also, the call to procedure sfp at the completion of dfs produces the fill-in corresponding to the pair (v_2, v_5) since $N_G W = \{v_2, v_5\}$ and $(v_2, v_5) \notin E$. Thus, for each independent clique of size 2 computed in G(U), exactly two fill-ins are produced in the Cholesky factor at the completion of procedure sfp. For the case where n is a multiple of 3, a simple analysis shows that $|S^*| = n/3$ and that $G(V - S^*)$ consists of $n/3 - 1$ cliques of size 2 and two interior cliques of size 1. Thus, at the completion of the first pass of the **while** loop, the size of the Schur complement is n/3, while the total number of fill-ins produced in the Cholesky factor is $2(n/3 - 1)$. Hence, using the parallel ordering algorithm pal, the height of the elimination tree is proportional to $\log_3 n$ which is much shorter than the height $\log_2 n$ obtained using the even–odd reduction (nested dissection) scheme.

Next we analyze the time complexity of the parallel ordering alggorithm. Procedure search computes the set of vertices S in time proportional to $|V| + |E|$ since each vertex in V is visited once and each edge in E is visited twice. Excluding the call to classify, the total time spent in dfs is proportional to $|V| + |E|$ since dfs uses the depth-first search method (Tarjan, 1972) for computing the connected components of the induced subgraph $G(V - S)$. For each connected component G(U) of $G(V - S)$ that is not a clique, procedure cliques(U) computes independent cliques in G(U) in time proportional to $|U| + |E(U)|$. For each independent clique G(W) of G(U), xfiils(W) updates the symbolic form of block X in time proportional to $|W| + |E(W)| + \nu(W)$, where $\nu(W)$ is the number of fill-ins produced in block X by the factorization of a submatrix of A corresponding to clique G(W). Also, we know that procedure sfp requires time proportional to $|V| + |E| + |E^*| - E(S^*)|$, and so the time spent in the first iteration of the **while** loop is proprtional to $|V| + |E| + \nu(V - S^*)$, where $\nu(V - S^*)$ is the total number of fill-ins produced by the symbolic factorization of pivot block A. Upon the completion of the first iteration of the **while** loop, vertex set V and edge set E are replaced by S^* and E^*, respectively, and the process is repeated. Thus, for small number of iterations of the **while** loop, the new ordering algorithm computes an ordering P and the symbolic Cholesky factor of PMP^T in time comparable to the best of commonly used methods.

In the rest of this section, we present computational results obtained from the application of the new ordering algorithm to the 21 problems listed in tables 1 and 2. Since the Schur complement $D - X^TX$ plays a central role in the parallel ordering algorithm pal, we first highlight the behavior of $D - X^TX$ at various iterations of the algorithm. Table 3 gives the size of the Schur complement in percentage of the size of the original matrix at each iteration of the **while** loop for all 21 problems. The iteration number corresponding to the last nonzero entry in each row defines the specific iteration at which the Schur complement becomes full for that problem. Thus, for problem seba in table 3, the Schur complement is full at the completion of the seventh iteration. Also, note that the size of the full Schur complement for seba is about 2 percent ofthe size of the original matrix at completion. For the

1138-by-1138 matrix in problem 1138_bus, the size of the Schur complement is about 1 percent of the size of the original matrix. It is worth noting that for 15 of the 21 problems in table 3, the size of the Schur complement is less than 10 percent of the size of the corresponding original matrix.

Table 3. Size of Schur complement in percentage of original matrix size.

	Identifier	**Iteration number in parallel_ordering**																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	bcsstk20	36	15	8	4	2																
2	nos1	54	25	9	3	1																
3	seba	26	14	10	5	4	3	2														
4	israel	59	46	35	28	26	23	22														
5	share1b	77	61	44	33	31	23	17														
6	can_292	65	47	39	33	29	19	16	10	8	6											
7	brandy	53	44	39	36	33	30	30	29	27	26	20										
8	685_bus	54	37	28	22	19	15	12	8	6	4	3	2									
9	bandm	61	47	36	32	30	29	27	23	21	16	13	11									
10	scfxm1	65	45	39	33	28	26	25	23	19	15	9	6									
11	1138_bus	41	20	11	7	5	4	3	3	2	2	2	1	1								
12	pores_3	57	36	28	21	16	14	13	11	9	8	5	3	2								
13	bcsstk19	64	54	45	41	38	31	29	24	20	14	10	7	3								
14	fffff800	69	56	48	41	32	30	26	25	23	21	18	17	15								
15	shell	47	31	24	19	16	14	13	12	10	9	9	8	8	4							
16	can_1054	65	54	44	39	34	28	25	23	21	17	13	9	8	8							
17	can_1072	65	53	44	39	36	31	28	25	23	21	17	14	11	9	3						
18	pores_2	57	46	42	38	35	33	31	29	27	26	22	20	16	13	8	5					
19	nos5	86	75	67	62	58	56	54	52	47	46	43	39	38	30	29	18					
20	pilot	62	45	35	30	28	26	24	23	20	19	17	15	14	13	11	11	8				
21	25fv47	63	51	45	40	35	33	31	30	28	27	26	24	22	21	19	15	14	14	12	9	9

Let T(i, j) denote the execution time at completion of jth iteration of the **while** loop for problem i, and let T(i, *) denote the overall time required by the new algorithm to compute permutation matrix P and symbolic Cholesky factor R of PMP^T for problem i. Also, let F(i, 1) and F(i, *) designate the sets of fill-ins produced at completion of the first iteration of the **while** loop and at completion of the algorithm, respectively. The first five columns in table 4 give T(i, 1) through T(i, 5) in percentage of the overall time T(i, *), while the sixth column gives the ratio |F(i, 1)|/|F(i, *)| for all 21 problems.

Let M be any sparse symmetric matrix and let G = (V, E) be the undirected graph of M. Then, by the time complexity analysis of the new algorithm, the time spent in the first iteration of the **while** loop is proportional to |V| + |E| + |F(i,1)| for the ith problem. Thus, for the case where |F(i, 1)| is close to |F(i, 1)|, the first iteration of the **while** loop has the complexity of an algorithm that computes an ordering P and the symbolic Cholesky factor R of PMP^T in linear time. Consequently, if we let b(i) designate the least integer greater than or equal to T(i, *)/T(i, 1), then the new algorithm requires at most b(i) applications of a linear-time algorithm to compute both P and R. Hence, for small values of b(i), the parallel ordering algorithm pal is comparable to a linear-time algorithm. However, it should be pointed out that b(i) is a loose upper bound if the ratio |F(i, 1)|/|F(i, *)| is small since the work

done in the first iteration of the **while** loop is taken to be comparable to a linear-time algorithm that accounts for all the fill-ins in F(i, *) and not only F(i, 1). Column 7 in table 4 gives the bound b(i) for all 21 problems.

Table 4. Execution times at completion of first five iterations in percentage of overall time.

	Identifier	Iteration number					|F(i,1)|/|F(i,*)|	b(i)
		1	2	3	4	5		
1	bcsstk20	60.55	82.12	89.30	96.60	100	.603	2
2	nos1	56.83	77.53	91.91	97.48	100	.762	2
3	seba	69.18	83.71	89.94	94.89	97.19	.751	2
4	israel	24.62	43.29	58.82	73.07	83.39	.642	5
5	share1b	21.09	44.45	58.89	72.04	82.62	.444	5
6	can_292	20.39	39.88	52.30	63.12	71.44	.320	5
7	brandy	15.50	24.53	33.69	42.92	51.50	.229	7
8	685_bus	26.78	43.01	54.17	63.33	71.26	.288	4
9	bandm	18.31	31.09	42.55	51.15	58.31	.148	6
10	scfxm1	15.69	27.45	37.00	47.37	55.82	.239	7
11	1138_bus	46.47	66.48	76.65	82.29	85.78	.371	3
12	pores_3	28.26	46.90	57.54	68.89	76.30	.232	4
13	bcsstk19	16.90	26.44	35.14	43.53	52.61	.286	6
14	fffff800	10.82	21.19	30.77	39.32	48.68	.150	10
15	shell	20.58	31.39	38.55	45.29	51.35	.208	5
16	can_1054	14.37	22.73	31.16	39.01	47.09	.205	7
17	can_1072	13.77	21.78	29.45	36.65	43.52	.201	8
18	pores_2	6.62	11.92	16.90	21.20	26.04	.078	16
19	nos5	2.15	4.92	8.13	11.82	15.90	.098	46
20	pilot	9.08	16.29	23.10	29.56	35.45	.133	11
21	25fv47	6.11	10.49	14.72	19.12	23.80	.086	17

The data included in column 7 of table 4 provides a number of worthwhile facts. First, for 16 of the 21 problems listed in table 4, the upper bound b(i) is less than or equal to 8. Also, for 12 of these 16 problems, |F(i, 1)| is strictly less than |F(i, *)|/2, which suggests that the integer 8 is a loose upper bound for these 12 problems. Second, for the remaining 5 of the 21 problems in table 4, the ratio |F(i, 1)|/|F(i, *)| is so small that the upper bounds computed for these problems are extremely loose. For example, the number of fill-ins computed in the first iteration of the **while** loop for problems pores_2, nos5, and 25fv47 is strictly less than 10 percent of the total number of fill-ins, and so the work done in the first iteration for these problems is much smaller than that required by an algorithm with time complexity proportional to |V| + |E| + |F(i, *)|. Third, the very small amount of work done in the first iteration of the **while** loop for problem nos5 correlates well with the very large Schur complement reported for nos5 in column 1 of table 3. In other words, the little work done in the first application of the **while** loop to problem nos5 is a result of the little parallelism identified by the new algorithm in the original problem. It is interesting to note that this behavior prevails at remaining iterations up to one before the final application of the **while** loop to nos5. For problems of this type, the new algorithm may not be well-suited since little parallelism is discovered per iteration.

We conclude our analysis of the 21 problems with a discussion centered on the distribution of workload associated with a vertex partition computed by the parallel ordering algorithm. An important concern in parallel computation is the proper distribution of workload across the processors of a parallel architecture computer. Let A be the pivot block associated with the vertex partition $\Pi^* = (V_1, V_2, \ldots, V_r, S^*)$ computed at a given iteration of the **while** loop. By construction, A is an r-by-r block diagonal matrix and so there exists no interprocessor communication between any two tasks resulting from the factorization of the r full diagonal blocks of A. However, since the vertex partition algorithm does not provide any control over the granularity of these r parallel tasks, workload imbalance may occur across the processors of a parallel machine. To deal with this workload balance problem, we describe in (Kevorkian, 1993) a strategy that uses the parameters $|V_1|$ through $|V_r|$ for decomposing the full diagonal blocks of A into smaller rectangular blocks.

Our experiments with the 21 problems listed in tables 1 and 2 have shown that for all pivot blocks constructed at the first seven iterations of the **while** loop all diagonal blocks ranged from 1 to 11 in size. For 15 of these 21 problems, the largest diagonal block was a 6-by-6 full matrix. Beyond seven iterations, the extent of parallelism detected by the new ordering algorithm is very small relative to the first few iterations. To quantify this observation, let r(i,j) designate the number of parallel tasks generated at jth iteration of **while** loop in percentage of the number of parallel tasks generated at the first iteration for problem i. Figure 6 gives min(r(i,j)), max(r(i,j)), and r(i,j) averaged over all i. By figure 6, the number of parallel tasks identified in a Schur complement beyond the sixth iteration is less than 8 percent of the number of parallel tasks in the original problem for all 21 problems considered in tables 1 and 2.

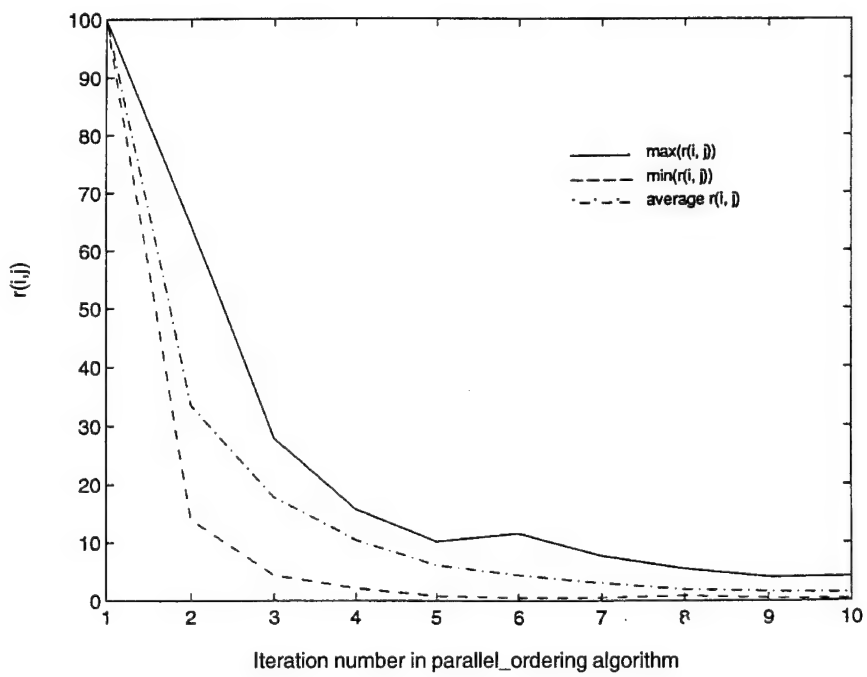


Figure 6. Number of parallel tasks per iteration in percent of number of parallel tasks at first iteration.

7. CONCLUSIONS

For a sparse symmetric matrix M, we have presented an ordering algorithm to compute a permutation matrix P so that parallelism inherent in M is fully exposed in PMP^T. The application of the new parallel ordering algorithm to a large set of sparse matrices taken from the Harwell–Boeing sparse matrix collection and industrial linear and quadratic programming problems show that this algorithm is an effective tool for exposing parallelism in arbitrary sparse symmetric matrices. Our experiments also show that the new parallel ordering algorithm compares favorably with a highly refined implementation of the minimum degree method in keeping the number of fill-ins and elimination tree heights small.

8. REFERENCES

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1976. The Design and Analysis of Computer Algorithms, Reading, MA, Addison-Wesley, 3rd Printing.

Dirac, G. A. 1961. On rigid circuit graphs, Abhandlungen aus dem Mathematischen Seminar der Universitat Hamburg, 25, pp. 70-76.

Duff, I. S., R. G. Grimes, and J. G. Lewis. 1989. Sparse matrix test problems, ACM Trans. Math. Software, 15, pp. 1-14.

George, A. and J. W.-H Liu. 1981. Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Inc., Engleweek Cliffs, New Hersey.

_____. 1989. The evolution of the minimum degree algorithm, SIAM Review, 31 (1989), pp. 1-19.

Gilbert, J. R., C. Moler, and R. Schreiber. 1992. Sparse Matrices in MATLAB: Design and implementation, SIAM J. Matrix Anal. Appl., 13, pp. 333–356.

Gill, P. E., W. Murray, D. B. Ponceleon, and M. A. Saunders. 1991. Solving reduced KKT systems in barrier methods for linear and quadratic programming, Technical Report SOL 91-7, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA.

_____. 1992. Preconditioners for indefinite systems arising in optimization, SIAM J. Matrix Anal. Appl., 13, pp. 292-311.

Heath, M. T., E. Ng, and B. W. Peyton. 1991. Parallel algorithms for sparse linear systems, SIAM Review, 33, pp. 420-460.

Jess, J. A. G. and H. G. M. Hees. 1982. A data structure for parallel L/U decomposition, IEEE Trans. Comput., 31, pp.231-239.

Kevorkian, A. K. 1993. Decomposition of large sparse symmetric systems for parallel computation. Part 1. Parallelization Tool Roadmap, NRaD Technical Report 1572, NCCOSC RDT&E Division, San Diego, CA 92152–5001.

Lekkerkerker, C. G. and J. CH. Boland. 1962. Representation of a finite graph by a set of intervals on the real line, Polska Akademia Nauk Fundamenta Mathematicae, LI, pp. 45-64.

Liu, J. W.-H. 1985. Modification of the minimum degree algorithm by multiple elimination, ACM Trans. Math. Software, 11, pp. 141-153.

_____. 1986. A compact row storage scheme for Cholesky factors using elimination trees, ACM Trans. Math. Software, 12, pp. 127-148.

_____. 1989. Reordering sparse matrices for parallel computation, Parallel Computing, 11, pp. 73-91.

The Mathworks. 1990. Pro-Matlab User's Guide, sough Natick, MA.

Parter, S. 1961. The use of linear graphs in Guass elimination, SIAM Review, 3, pp. 119-130.

Pothen, A., H. D. Simon, and L. Wang. 1992. Spectral nested dissection, Technical Report No. RNR-92-003, Nasa Ames Research Center.

Rose, D. J., R. E. Tarjan, and G. S. Lueker. 1976. Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput., 5, pp. 266-283.

Schreiber, R. 1982. A new implementation of the sparse Guassian elimination, ACM Trans. Math. Software, 8, pp. 256-276.

Tarjan, R. E. 1972. Depth-first search and linear graph algorithms, SIAM J. Comput., 1, pp. 146-160.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1995	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
AN ORDERING ALGORITHM FOR EXPOSING PARALLELISM IN SPARSE SYMMETRIC MATRICES	AN: DN302038 PE: 0601152N PROJ: ZW62
6. AUTHOR(S) Aram K. Kevorkian	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E Division San Diego, CA 92152–5001	8. PERFORMING ORGANIZATION REPORT NUMBER TR 1697
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Chief of Naval Research OCNR–10P Arlington, VA 22217–5000	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Authorized for public release; distribution is unlimited.	

13. ABSTRACT *(Maximum 200 words)*

Given a sparse symmetric matrix M, we develop an ordering algorithm to find a permutation matrix P so that parallelism inherent in M is fully exposed in the matrix PMP^T. The key steps in the ordering algorithm are as follows. First, compute in the undirected graph G = (V, E) of M a set of vertices S* such that the induced subgraph G(V - S*) contains parallel regions inherent in M. This step gives rise to a block diagonal matrix A such that each diagonal block is a full matrix in M. Second, factor block diagonal matrix A symbolically and compute the symbolic form of the Schur complement of A in M. Third, replace original matrix M by the symbolic Schur complement, and repeat the process until the symbolic Schur complement is a full matrix. By a property of the set S*, the ordering algorithm takes advantage of all principal submatrices of M that do not produce fill-in in any part of M when symbolically factored. Applications to a large collection of sparse matrices show that the new ordering algorithm is an effective tool for exposing parallelism in sparse symmetric problems. Also, comparisons show that the new algorithm fares favorably with commonly used ordering methods.

14. SUBJECT TERMS	15. NUMBER OF PAGES
cliques, elimination tree, parallel computation, separators, simplicial vertices, vertex partition	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
A. K. Kevorkian	(619) 553-2058	Code 7801